EXHIBIT B

EXHIBIT B

XMC Motion Control

XMCAPI Reference

Revision:	Third Draft
Author:	Dave Brown
Date:	Wednesday, Feb 22, 1995
Description:	This document is the XMCAPI reference describing all functions, standard OLE interfaces, and custom OLE interfaces making up the XMCAPI.

Revision History:	4/15/94 (DB)	- First Draft: Initial writing.
	7/1/94 (DB)	- Second Draft: Split design off small business plan, incorporated suggestions.
	2/22/95 (DB)	- Third Draft - Split XMCAPI reference off design guide.

This is a preliminary release of the documentation. It may be changed substantially prior to final commercial release. This document is provided for discussion purposes only in strict confidence and is governed by the Don-Disclosure or Beta Agreement that you or a representative from your company has signed.

ROY-G-BIV Corporation makes no warranties on the information contained within this document or software. The entire risk of using this document or software including the results of its use, remains with the user. No part of this document may be reproduced without express written permission from ROY-G-BIV Corporation.

ROY-G-BIV, XMC, XMCAPI, and XMCSPI are trademarks of ROY-G-BIV Corporation.

Microsoft, Visual C++, Visual Basic, WIN32, and Microsoft Excel 5.0 are registered trademarks and Windows, Windows NT, MFC, WIN32s, and OLE are trademarks of Microsoft Corporation.

Borland and Borland C++ are trademarks of Borland International, Inc.

Printed in the United States of America.

Table of Contents

1.0 Overview

To use XMC, applications call the methods defined by the XMCAPI OLE interfaces to perform the motion control services needed. The full XMCAPI comprises a set of OLE interfaces corresponding with common objects found in a motion control system. For example, a motion control system may contain servo and stepper motors, encoders, limit switches, and a joystick. Within the XMCAPI are interfaces corresponding to each of these objects. The full XMCAPI contains implementations of both standard OLE 2.0 interfaces and a rich set of custom OLE interfaces designed specifically for motion control.

This reference manual describes all OLE interfaces in the XMCAPI. Each section discussing an interfaces starts with a description of the interface and how it is used. Next, all functions in the interface are listed by catagory.

Chapter **2.0 Standard OLE Interfaces**, describes all standard interfaces exposed in the XMCAPI. Next, Chapter **3.0 XMC Driver Administrator**, discusses both the standard and custom interfaces exposed by the Driver Administrator that make up a small portion of the XMCAPI. Finally, Chapter **4.0 XMC Motion Control Component**, discusses the standard and custom interfaces exposed by the Motion Control Component. The Motion Control Component implements and exposes the majority of the XMCAPI. Together, the Motion Control Component and the Driver Administrator implement and expose the full set of interfaces defined by the XMCAPI.

2.0 Standard OLE Interfaces

Several standard OLE interfaces must be supported when following the OLE software model. Both the **IUnknown** and **IClassFactory** interfaces are examples of required interfaces. The **IDataObject** interface is used when transferring or caching data. The **IDispatch** is provided to support OLE Automation between processes. This section describes all standard OLE interfaces used by the Driver Administrator, Motion Control Component, and Motion Control Drivers.

2.1 IClassFactory Interface

The **IClassFactory** interface is used to create an instance of an object and return to the caller a pointer to an interface.

General

```
(*pCF)->CreateInstance()
(*pCF)->LockServer()
```

2.2 IDispatch Interface

The **IDispatch** interface is used to support OLE Automation. Normally, this interface is used by the OLE Automation engine, not the C/C++ application developer.

General

```
(*pDisp)->GetIDsOfNames()
(*pDisp)->GetTypeInfo()
(*pDisp)->GetTypeInfoCount()
(*pDisp)->Invoke()
```

2.3 IEnumX Interface

The **IEnumX** interface is a generic OLE interface that other enumeration interfaces always inherit from. This interface is used to create and move through an enumeration of data. The **IXMC_EnumInterface** interface inherits from **IEnumX** to implement the interface support querying functionality.

General

```
(*pEnum)->Next()
(*pEnum)->Skip()
(*pEnum)->Reset()
(*pEnum)->Clone()
```

2.4 IUnknown Interface

The **IUnknown** interface is a standard OLE interface exposed by all OLE objects. In a way it is a common base interface.

General

```
(*pUnk)->QueryInterface()
(*pUnk)->AddRef()
(*pUnk)->Release()
```

3.0 XMC Driver Administrator

The Driver Administrator, which implements a small subset of the XMCAPI set of functions, is used by both the user and the Motion Control Component. It is used by the user to add new Motion Control Drivers, configure existing drivers, and remove old ones. Later when an application uses the Motion Control Component, the component queries the Driver Administrator for the current driver to use.

NOTE: OLE 2.0 interfaces are used to implement all functions in the XMCAPI.

The following OLE 2.0 interfaces are custom OLE interfaces in the XMCAPI supported by the Driver Administrator.

```
IXMC_DriverAdmin
```

3.1 Standard OLE Interfaces

The following standard OLE interfaces are supported by the Driver Administrator. For more information on each interface, see section *6.0 Standard OLE Interfaces*.

```
IClassFactory
IDispatch
IUnknown
```

3.2 Custom OLE Interfaces

The following custom interface is a part of the XMCAPI and an extension to the OLE 2.0 software model.

3.2.1 IXMC_DriverAdmin

This interface is used to manipulate and query the Driver Administrator settings. The Driver Administrator application, run by the user, manipulates the administrator's settings. And, the Motion Control Component queries the administrator for a pointer to the current driver to use.

Querying Attributes

```
(*pXMCDrvAdmin)->GetSelectedDrivers()
(*pXMCDrvAdmin)->IsDiagnosticTestingOn()
(*pXMCDrvAdmin)->IsAPILoggingOn()
(*pXMCDrvAdmin)->IsCmdLoggingOn()
(*pXMCDrvAdmin)->IsBitLoggingOn()
```

Setting Attributes

```
(*pXMCDrvAdmin)->SetAPIStream()
(*pXMCDrvAdmin)->SetCmdStream()
(*pXMCDrvAdmin)->SetBitStream()
```

Actions

```
(*pXMCDrvAdmin)->EnableAPILogging()
(*pXMCDrvAdmin)->EnableCmdLogging()
(*pXMCDrvAdmin)->EnableBitLogging()
(*pXMCDrvAdmin)->EnableDiagnostics()
(*pXMCDrvAdmin)->ViewInterfaceSupport()
```

3.2.1 IXMC_EnumDriver

The IXMC_EnumDriver interface is used by the XMC Motion Control Component when building the interface enumeration describing the level of XMC support provided by each driver. This interface inherits from the standard IEnumX OLE interface.

4.0 XMC Motion Control Component

The Motion Control Component, which implements the majority of the XMCAPI set of functions, is used by applications. As a complete set, the OLE interfaces in the XMCAPI are separated into the four catagories below.

1. **Standard** - Standard OLE interfaces.
2. **Motion Control** - Custom OLE interfaces used for motion control and code generation.
3. **Code Generation** - Custom OLE interfaces used for code generation only.
4. **Diagnostic** - Custom OLE interfaces used for debugging the system.

XMCAPI interfaces follow the standard OLE 2.0 COM Object software model. The following OLE 2.0 interfaces make up the XMCAPI set of functions implemented by the Motion Control Component:

Standard
- IClassFactory
- IDataObject
- IDispatch
- IEnumX
- IUnknown

Motion Control
- IXMC_Counter
- IXMC_Display
- IXMC_DynamicState
- IXMC_Encoder
- IXMC_EnumInterface
- IXMC_IO
- IXMC_Joystick
- IXMC_Limits
- IXMC_Motion
- IXMC_StaticState
- IXMC_Timer

Code Generation
- IXMC_Operator
- IXMC_Program
- IXMC_ProgramFlow
- IXMC_ProgramMgmt
- IXMC_Subroutine
- IXMC_Variable

Diagnostic
- IXMC_Error
- IXMC_Debug

4.1 Standard OLE Interfaces

The following standard OLE interfaces are supported by the Motion Control Component. For more information on each interface, see section *6.0 Standard OLE Interfaces*.

- IClassFactory
- IDataObject
- IDispatch
- IEnumX
- IUnknown

4.2 Custom OLE Interfaces - Motion Control

The general set of interfaces is made up from all main motion control interfaces. Most interfaces resemble objects found within the motion control system.

4.2.1 IXMC_Counter Interface

The IXMC_Counter interface allows the programmer to manipulate a hardware counter.

Actions

```
(*pCounter)->Advise()
(*pCounter)->AttachEncoder()
(*pCounter)->DetachEncoder()
(*pCounter)->Enable()
(*pCounter)->Reset()
(*pCounter)->UnAdvise()
```

Querying Attributes

```
(*pCounter)->GetCount()
```

4.2.2 IXMC_DynamicState Interface

The IXMC_DynamicState interface manages high level system tasks, such as initialization, and allows the user to get all state information describing the motion control system. When queried, other interfaces use an internal pointer to the IXMC_DynamicState interface to retrieve state values, which are returned to the caller. The following methods make up the IXMC_DynamicState interface:

Configuration

```
(*pDynSt)->InitializeEx()
(*pDynSt)->Intialize()
(*pDynSt)->Reset()
(*pDynSt)->ShutDown()
```

Querying Attributes

```
(*pDynSt)->AreHardwareLimitsOn()
(*pDynSt)->AreSoftwareLimitsOn()
(*pDynSt)->GetAcceleration()
(*pDynSt)->GetActualPosition()
(*pDynSt)->GetActualVelocity()
(*pDynSt)->GetCommandedPosition()
(*pDynSt)->GetCommandedVelocity()
(*pDynSt)->GetCount()
(*pDynSt)->GetDeceleration()
(*pDynSt)->GetErrorStatus()
(*pDynSt)->GetHomePosition()
(*pDynSt)->GetEncoderResolution()
(*pDynSt)->GetJogVelocityHigh()
(*pDynSt)->GetJogVelocityLow()
(*pDynSt)->GetJoystickVelocityHigh()
(*pDynSt)->GetJoystickVelocityLow()
(*pDynSt)->GetJoystickAcceleration()
(*pDynSt)->GetJoystickDeceleration()
(*pDynSt)->GetLimitDeceleration()
(*pDynSt)->GetSoftwareLimitPositions()
(*pDynSt)->GetMapMode()
(*pDynSt)->GetPosition()
(*pDynSt)->GetTimerTick()
(*pDynSt)->GetUnits()
(*pDynSt)->GetVelocity()
(*pDynSt)->GetZeroPosition()
(*pDynSt)->IsCounterOn()
(*pDynSt)->IsInMotion()
(*pDynSt)->IsInterpolationOn()
```

```
(*pDynSt)->IsJoystickOn()
(*pDynSt)->IsMotionOn()
(*pDynSt)->IsPositionErrorCorrectionOn()
(*pDynSt)->IsTimerOn()
(*pDynSt)->IsIOOutputOn()
(*pDynSt)->IsIOInputOn()
```

Setting Attributes

```
(*pDynSt)->ClearErrors()
```

Actions

```
(*pDynSt)->GetCS()
(*pDynSt)->ReleaseCS()
```

4.2.3 IXMC_Encoder Interface

The **IXMC_Encoder** interface is used when working with encoder hardware. Once initialized, a pointer to an **IXMC_Encoder** interface may be attached to an **IXMC_Motion** interface which allows them to work together. Once attached, the **IXMC_Motion** interface uses the **IXMC_Encoder** interface position related functions, like setting the home position or querying the current position.

Querying Attributes

```
(*pEncoder)->GetErrorStatus()
(*pEncoder)->GetHomePosition()
(*pEncoder)->GetPosition()
(*pEncoder)->GetZeroPosition()
(*pEncoder)->GetResolution()
```

Setting Attributes

```
(*pEncoder)->SetHomePosition()
(*pEncoder)->SetHomePositionEx()
(*pEncoder)->SetZeroPosition()
(*pEncoder)->SetZeroPositionEx()
(*pEncoder)->SetResolution()
```

4.2.4 IXMC_EnumInterface Interface

The **IXMC_EnumInterface** interface is used to access the list of the XMCAPI interface names and a description of their implementations. For example, if the motion control hardware or software driver controlling the hardware do not support code generation, all code generation interfaces in the enumeration would be listed as "NOT IMPLEMENTED". Knowing what interfaces are and are not supported allow particular programs to decide what drivers are actually capable of running with the application. This interface inherits from the standard **IEnumX** OLE interface.

Inheritance

```
IEnumX
   |
   |--- IXMC_EnumInterface
```

4.2.5 IXMC_IO Interface

This interface is used by applications to manipulate both digital and analog I/O support located on the motion control hardware. The following methods are available from the Motion Component in the IXMC_IO interface:

Configuration

```
(*pIO)->Initialize()
(*pIO)->InitializeEx()
```

Querying Attributes

```
(*pIO)->GetErrorStatus()
(*pIO)->IsIntputEnabled()
(*pIO)->IsOutputEnabled()
```

Actions

```
(*pIO)->Advise()
(*pIO)->AttachMotion()
(*pIO)->DetachMotion()
(*pIO)->EnableInput()
(*pIO)->EnableOutput()
(*pIO)->Read()
(*pIO)->UnAdvise()
(*pIO)->Write()
```

4.2.6 IXMC_Joystick Interface

The IXMC_Joystick interface is used to manipulate a hardware joystick. When the joystick is enabled, all instances of the IXMC_Motion interface will be disabled.

Querying Attributes

```
(*pJS)->GetAcceleration()
(*pJS)->GetDeceleration()
(*pJS)->GetErrorStatus()
(*pJS)->GetVelocityHigh()
(*pJS)->GetVelocityLow()
(*pJS)->IsEnabled()
```

Setting Attributes

```
(*pJS)->SetAcceleration()
(*pJS)->SetDeceleration()
(*pJS)->SetVelocityHigh()
(*pJS)->SetVelocityLow()
(*pJS)->SetZero()
```

Actions

```
(*pJS)->AttachAnalogIO()
(*pJS)->AttachLimits()
(*pJS)->DetachAnalogIO()
(*pJS)->DetachLimits()
(*pJS)->Enable()
```

4.2.7 IXMC_Limits Interface

The IXMC_Limits interface is used to set software limit positions. The IXMC_Limits, IXMC_Joystick, and IXMC_Motion interfaces work together to implement the software limits.

Querying Attributes

```
(*pLim)->GetErrorStatus()
(*pLim)->GetLimitPositions()
(*pLim)->GetDeceleration()
(*pLim)->AreSoftwareLimitsOn()
```

```
(*pLim)->AreHardwareLimitsOn()
```

Setting Attributes

```
(*pLim)->SetLimitPositions()
(*pLim)->SetDeceleration()
```

Actions

```
(*pLim)->EnableSoftwareLimits()
(*pLim)->EnableHardwareLimits()
```

4.2.8 IXMC_Motion Interface

The IXMC_Motion interface is used to perform movement operations. Most functions are passed an array of AXIS_DATA specifying the appropriate axis data.

Configuration

```
(*pMotion)->Initialize()
(*pMotion)->Tune()
(*pMotion)->TuneEx()
```

Querying Attributes

```
(*pMotion)->GetAcceleration()
(*pMotion)->GetDeceleration()
(*pMotion)->GetErrorStatus()
(*pMotion)->GetVelocity()
(*pMotion)->GetPosition()
(*pMotion)->GetUnits()
(*pMotion)->IsMotionOn()
(*pMotion)->IsInterpolationOn()
```

Setting Attributes

```
(*pMotion)->AttachEncoder()
(*pMotion)->AttachLimits()
(*pMotion)->DetachEncoder()
(*pMotion)->DetachLimits()
(*pMotion)->SetAcceleration()
(*pMotion)->SetDeceleration()
(*pMotion)->SetPosition()
(*pMotion)->SetVelocity()
(*pMotion)->SetJogVelocityHigh()
(*pMotion)->SetJogVelocityLow()
```

Actions

```
(*pMotion)->Advise()
(*pMotion)->CreatePath()
(*pMotion)->EnableInterpolation()
(*pMotion)->EnableMotion()
(*pMotion)->EnablePositionErrorCorrection()
(*pMotion)->GoHome()
(*pMotion)->GoZero()
(*pMotion)->Jog()
(*pMotion)->Kill()
(*pMotion)->MoveAbs()
(*pMotion)->MoveContinous()
(*pMotion)->MoveRes()
(*pMotion)->Stop()
(*pMotion)->TraceArc()
(*pMotion)->TracePath()
(*pMotion)->UnAdvise()
```

4.2.9 IXMC_StaticState Interface

The IXMC_StaticState interface manages all static data in the motion control system, such as scaling factors, maximum velocities and maximum accelerations. The following methods make up the IXMC_StaticState interface:

Configuration

```
(*pStatSt)->InitializeEx()
(*pStatSt)->Intialize()
```

Querying Attributes

```
(*pStatSt)->GetAxisScaling()
(*pStatSt)->GetMaxAcceleration()
(*pStatSt)->GetMaxDeceleration()
(*pStatSt)->GetMaxVelocity()
(*pStatSt)->GetPathScaling()
(*pStatSt)->GetOperatingMode()
```

Setting Attributes

```
(*pStatSt)->SetAxisScaling()
(*pStatSt)->SetMaxAcceleration()
(*pStatSt)->SetMaxDeceleration()
(*pStatSt)->SetMaxVelocity()
(*pStatSt)->SetPathScaling()
(*pStatSt)->SetOperatingMode()
```

Actions

```
(*pStatSt)->CommitState()
(*pStatSt)->EnableScalingFactors()
(*pStatSt)->GetState()
(*pStatSt)->ReleaseState()
```

4.2.10 IXMC_Timer Interface

The IXMC_Timer interface allows the programmer to manipulate a hardware timer.

Querying Attributes

```
(*pTimer)->GetDelay()
(*pTimer)->GetErrorStatus()
(*pTimer)->GetElapsedTime()
(*pTimer)->GetResolution()
(*pTimer)->IsInDelay()
```

Setting Attributes

```
(*pTimer)->SetDelay()
```

Actions

```
(*pTimer)->Advise()
(*pTimer)->DoDelay()
(*pTimer)->Enable()
(*pTimer)->Reset()
(*pTimer)->Start()
(*pTimer)->Stop()
(*pTimer)->UnAdvise()
```

4.3 Custom OLE Interfaces - Code Generation

Unlike the set of general interfaces, that may either run in either real-time or generating code modes, the code generation interfaces are only used when generating code. When used in conjunction with other general interfaces, complete motion control programs may be generated. This section discusses each of the code generation interfaces.

4.3.1 IXMC_Operator Interface

The IXMC_Operator interface contains general program statement generating functions including bitwise operations, mathematical operations, logical operations, and general conditional operations. The following functions are in the IXMC_Operator interface:

Constants

```
CE_PI
```

Bitwise

```
(*pOper)->BitAnd()
(*pOper)->BitNot()
(*pOper)->BitOr()
(*pOper)->BitShiftL()
(*pOper)->BitShiftR()
(*pOper)->BitXOr()
```

Logical

```
(*pOper)->And()
(*pOper)->Not()
(*pOper)->Or()
```

Mathematical

```
(*pOper)->Add()
(*pOper)->Bit()
(*pOper)->Div()
(*pOper)->LeftParen()
(*pOper)->Mult()
(*pOper)->RightParen()
(*pOper)->Sqrt()
(*pOper)->Sub()
```

Trigonometric

```
(*pOper)->ACos()
(*pOper)->ASin()
(*pOper)->ATan()
(*pOper)->Cos()
(*pOper)->Sin()
(*pOper)->Tan()
```

Labels/Strings/Comments

```
(*pOper)->ASCIIChar()
(*pOper)->Comment()
(*pOper)->DeclLabel()
(*pOper)->String()
```

Conditional

```
(*pOper)->Equal()
(*pOper)->GreaterThan()
(*pOper)->GreaterThanEqual()
(*pOper)->LessThan()
(*pOper)->LessThanEqual()
(*pOper)->NotEqual()
```

4.3.2 IXMC_Program Interface

The IXMC_Program interface is used to generate the general program shell. Several other program flow functions like causing the program to sleep for a specified amount of time are also included. The following methods make up the interface:

Program Flow

```
(*pProg)->Combine()
(*pProg)->Define()
(*pProg)->Download()
```

```
(*pProg)->End()
(*pProg)->Exit()
(*pProg)->RenderAsText()
(*pProg)->Run()
(*pProg)->Sleep()
(*pProg)->Stop()
```

4.3.3 IXMC_ProgramFlow Interface

The IXMC_ProgramFlow interface is used to generate code used to control the flow of a program. Methods are available to generate if-else and general looping statements. The following methods are in the IXMC_ProgramFlow interface:

If/Else

```
(*pProgFlow)->IfOpen()
(*pProgFlow)->IfClose()
(*pProgFlow)->Else()
(*pProgFlow)->EndIf()
```

While

```
(*pProgFlow)->WhileOpen()
(*pProgFlow)->WhileClose()
(*pProgFlow)->EndWhile()
```

Repeat

```
(*pProgFlow)->Repeat()
(*pProgFlow)->UntilOpen()
(*pProgFlow)->UntilClose()
```

Loops

```
(*pProgFlow)->Loop()
(*pProgFlow)->EndLoop()
```

Misc.

```
(*PProgFlow)->Break()
(*pProgFlow)->GoTo()
```

4.3.4 IXMC_ProgramMgmt Interface

The IXMC_ProgramMgmt interface is used when working with programs on a high level. Running and saving programs are a few operations included in this interface. The following methods make up the interface:

Program Management

```
(*pProgMgmt)->Download()
(*pProgMgmt)->Upload()
(*pProgMgmt)->SetOutput()
(*pProgMgmt)->OpenOutput()
(*pProgMgmt)->CloseOutput()
(*pProgMgmt)->FlushOutput()
```

4.3.5 IXMC_Subroutine Interface

The IXMC_Subroutine interface is used to generate and subroutines and calls to subroutines generated. The following methods are in the IXMC_Subroutine interface:

General

```
(*pSub)->Define()
(*pSub)->End()
(*pSub)->Call()
```

4.3.6 IXMC_Variable Interface

The IXMC_Variable interface is used to define and assign values to variables. The following methods are included in the interface:

Definition

```
(*pVar)->Define()
```

Assignment

```
(*pVar)->AssignNumeric()
(*pVar)->AssignBinary()
(*pVar)->AssignString()
```

4.4 Custom OLE Interfaces - Diagnostic

Diagnostic interfaces may be used by the application developer when debugging the system. The methods in the interface provide the user with logs of information generated when running the XMC software model. Logs may be created for XMCAPI calls, and all information flowing in and out of specified ports.

4.4.1 IXMC_Error Interface

The IXMC_Error interface is provided to give the application developer a consistant method of displaying and logging all XMC errors based on the HRESULT value returned by functions that fail. The following methods are in the interface:

Setting Attributes

```
(*pError)->SetOutput()
```

Actions

```
(*pError)->Display()
(*pError)->Log()
(*pError)->StringFromHRESULT()
```

4.4.2 IXMC_Debug Interface

The IXMC_Debug interface gives the application developer control over several log files that may be generated to help debug an application using the XMC software model. The following methods are in the interface:

Setting Attributes

```
(*pDebug)->SetXMCAPILogFile()
(*pDebug)->SetXMCAPIStream()
```

Actions

```
(*pDebug)->EnableXMCAPILogging()
(*pDebug)->EnableDiagnostics()
```

Appendix A

A.1 Reviewer Feedback

All feedback is important in order to evolve the specification into a standard software model for motion control hardware. Please, at a minimum, respond to the following issues:

1. Which areas, ideas, or concepts within the specification seem confusing to you?
2. Do you see any unresolved issues, other than those listed above?
3. Are there any main features missing from the specification that you would find useful?
4. Are there any specific scenario-maps you would like to see in the next revision?
5. Are there any specific C or C++ code examples you would like to see?
6. Are there any specific Visual Basic code examples you would like to see?

A.1.1 Correspondence

If possible, please send all feedback over email to the following address:

Dave Brown
ROY-G-BIV Corporation
Compuserve: 72103,2235
Internet: 72103.2235@compuserve.com

Otherwise, if you do not have access to email, please mail all correspondence to the following address:

Dave Brown
ROY-G-BIV Corporation
P.O. Box 1278
White Salmon, WA. 98672